U.S. DEPARTMENT OF COMMERCE • National Bureau of Standards

NBSIR 87-3627

Institute for Computer Sciences and Technology

Performance Measurement Instrumentation for Multiprocessor Computers

Robert J. Carpenter

U. S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Advanced Systems Division
Gaithersburg, MD 20899

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

August 1987

Partially sponsored by the
Defense Advanced Research Projects Agency
under ARPA order number 5520,
July 23, 1985, and July 28, 1986.

PERFORMANCE MEASUREMENT INSTRUMENTATION FOR MULTIPROCESSOR COMPUTERS

Robert J. Carpenter

Advanced Systems Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Work reported in this report was partially supported by the
Strategic Computing Initiative
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209
ARPA Order No. 5520, July 23, 1985,
as amended July 28, 1986.

U.S. Department of Commerce, Clarence Brown, Jr., Acting Secretary

National Bureau of Standards, Ernest Ambler, Director

August 1987

TABLE OF CONTENTS

Page

PERFORMANCE MEASUREMENT INSTRUMENTATION FOR MULTIPROCESSOR COMPUTERS

Robert J. Carpenter

The complexity of achieving near-optimum performance from multiprocessor parallel computers demonstrates a need for performance measurement. However, when multiple processors are acting in concert on a single problem, perturbations caused by measurement can be unacceptable. Additional hardware can reduce the perturbation caused by measurement, and can be offered in several stages of refinement and cost. The hardware can often be offered as an option; it is necessary to provide access to the required signals in the system's original design.

Key words: Performance measurement; Hardware instrumentation; Measurement equipment; Multiprocessor computers.

INTRODUCTION

Multiprocessor computers have been built (or proposed) with a wide range of architectures. When the goal of using multiple processors is *speedup*, getting a single job done faster, there must be a reasonable match between the algorithms, the data set, and both the architecture and its implementation. The quality of this match is hard to determine *a priori*, even for specific instances of hardware and software. If the match is poor, the result may be **very** poor performance, even slower than a uniprocessor. While the machine is a *given* to the computer user, algorithms and programming paradigms are not. Thus the user needs to be able to measure the response of the computer to various paradigms, test programs, and the eventually chosen algorithmic solution. The designer, configurer, and selector have more flexibility in tailoring the hardware and system software, but there is a common thread of need to avoid performance-destroying bottlenecks in the design, implementation and configuration. Almost all multiprocessor parallel computers can be configured in a wide range of size and cost; which configuration is best for a particular situation? The answer requires, of course, that performance measurement techniques be made available to the user. However, the comprehensive measurement of complex machines is extremely difficult, and in existing multiprocessor computers little or no effort has been made to facilitate the connection of measurement hardware to allow observation and measurement of the internal operations which are critical to performance. Performance measurement is often an *ad hoc*, one-time affair done during system design and debugging, but the necessary "hooks" are taken out before manufacture.

There is a trade-off between cost, accuracy and completeness of measurement. **Cost** can be monetary, physical size, or reduced performance caused by the mere presence of the measurement system, or the "hooks" for it. **Accuracy** includes both the errors caused by perturbation of the process being measured and the degree to which the (perturbed) operation is correctly quantified. **Completeness** is the degree to which the measurer's needs are met by the measurement process.

This preliminary discussion surveys a range of approaches to performance measurement, with examples of the machine connections needed for measurement without incurring excessive perturbation. Other approaches are possible, for example the SySM system from Harris for the Concert multiprocessor [MIT86][WHI86]. As will be discussed later, attempts to measure multiprocessor parallel computers can create so much perturbation to normal operation as to render the results useless. Data, once obtained, must be analyzed and presented to the user. Though important, this is not discussed, nor is testing for fault-tolerance and maintenance purposes.

NUMBERS AND MEASUREMENT OF PERFORMANCE

The collection of numbers (bus occupancy, cache hits, instructions per second) does not, alone, amount to measurement of a computer system. These numbers are obtained while the system is executing programs (a stimulus). In parallel systems, the numbers (results) can differ dramatically with different stimuli. A key aspect of measurement is the provision of well-characterized test routines (benchmarks) for use as the stimulus. The numbers obtained during measurement are useful only to the extent that they can be related to expected performance on a real application. Thus some relationship between the test routines and real applications must be known or developed. Small, kernel test routines characteristic of operations central to parallel execution are the most portable type of test stimuli. Results on these small routines can tell a lot about the performance of a machine; these results are vital in revealing the best programming style to use for the expensive task of moving much larger benchmarks to the machine in question [LYO87].

WHAT TO MEASURE?

"How long did it take?" is the most obvious question asked about the performance of a computer system doing a specific task. But, once that question can be answered without inducing significant experimental perturbations, there remains the question of "How can this be improved?" So - the characterization of a computer system involves

- Measurement of the time required to execute the test software, and
- Measurement of the utilization of the computing resources of the system to discover bottlenecks which may be corrected by hardware redesign, system reconfiguration, system expansion, use of another algorithm or a different programming paradigm, ...

These measurements are most useful if they can resolve chosen segments of the test software. Gross overall execution measurements give little insight on the best

corrective action to take.

COST CONSIDERATIONS IN MEASUREMENT

A number of possible measurement approaches will be presented. An attractive goal (for manufacturers) would be to offer the measurement hardware as an option, with essentially no incremental hardware cost to purchasers who did not want the measurement option. As will be seen, some measurement approaches can meet this criterion, while others emphatically cannot. Measurement data analysis and presentation software is clearly not needed by owners who don't buy the system measurement features. There may also be a system software cost in providing for measurement. System software such as compilers, linkers, and loaders may need to be modified to handle measurement pseudo-instructions, and to generate control programs for the measurement hardware.

MEASURING EXECUTION DURATION

There are two critical *events* to the measurement of execution duration; starting time and ending time (actually the difference between them). If the execution duration is long enough, the user may be able to use the timing service provided by the operating system without significantly perturbing the results. Unfortunately the requirement for long execution duration usually means that either the test software is very large and hard to characterize, or that a very unrealistic synthetic test routine has been created. Prior state can have a large effect on computer performance. Continual looping can result in most of the critical items being in cache memory, often an artificial situation. On the other hand, addition of code to "precondition" the machine (cache) state will grossly perturb overall execution time. Thus one needs to be able to insert time-measurement *events* freely, and without significant perturbation to the system under test. It is important to provide the user with a means to start and stop data taking (perhaps by special *events*; see below) to avoid taking data while the desired starting state is being recreated. Each time one resorts to the timing service of an operating system, perhaps 500 extra instructions are executed. The tolerance of computations to time-gathering perturbations of this sort can be divided into two classes: computations with one instruction stream, and those with multiple (simultaneously executing) instruction streams or with time-critical external relationships.

Systems with a Single Instruction Stream. If there is but one instruction stream and no machine state to upset (no pipelining, no fetch-ahead, no cache), and no external time-critical interactions, execution can be suspended at any time by an operating system call without great perturbation of execution characteristics. This implies no time-critical user or mass-storage I/O. Thus measurement can be accomplished by recourse to operating system services, and little hardware measurement support is required. This observation applies to both single- and multiple-data-stream processors with a

single instruction stream (SIMD as well as SISD).

Systems with Multiple Instruction Streams. All other computer systems (including "single-instruction-stream" machines with real-time I/O or separate I/O processors) are effectively multiple-instruction-stream machines, wherein the time relationships between the various instruction paths must be maintained. Individual calls to an operating system for measurement services would delay one processor, while allowing the others to proceed - seriously perturbing execution. The amount of perturbation which can be tolerated is related to the program *granule* size; how much computation is done between times that processes must interact and measurements must be taken. In general, the longer this time, the more perturbation that can be tolerated. One approach is to assign measurement as the sole task of one of the processors; surely a major distortion of system resources. These MIMD and real-time systems require at least some hardware support to obtain reliable measurement [SCH83]. This hardware for the support of measurement, and the connections it requires will be discussed below.

For the person who wants to measure an architecture and configuration, but not a particular implementation technology, timing results should be stated in terms of elapsed computer system clock cycles, not microseconds. System selection and real-time applications require results in terms of time.

Identifying Time Events in Test Program Execution

Events in the execution of a program can be identified in two ways: by inserting extra instructions in the software to explicitly trigger data capture, or by recognizing use of a specific instruction or data (address or value) by passive monitoring and pattern matching. The recognition of a user-specified event during test program execution generates a *trigger* to cause capture of the current time and other measurement data.

Inserted-Code Triggering. In some situations, the perturbation from extra measurement instructions may be tolerable [SCH83]. On the other hand, the perturbation may be insidious. Instruction fetch-ahead queues will be affected, caches will be perturbed, and memory management translation look-aside buffer entries may be affected. Inserted-code event triggering should only be considered for measurement events separated by at least some dozens of instructions in the same instruction path.

The advantages of inserted-code triggering are substantial. Child processes can easily identify themselves, as can processes sharing code. There is never false triggering such as might occur in an address-monitor trigger when instructions or data are fetched but never used. It is relatively easy to identify processor-process association. In tightly-coupled systems, hardware support which reduces the perturbation of measurement can be accomplished centrally, at relatively low cost.

Pattern-Matching Triggering. Time-measurement events can be recognized without any program perturbation by use of a pattern-sensitive monitoring system. This relatively-complex trigger hardware is designed to recognize certain instruction or data addresses or values and must usually be replicated for each processor in the system. While this approach has the great advantage of being non-perturbing, it suffers practical operational problems in addition to high cost. The first problem is that the desired signals may not be easy to reach; they may be buried inside a VLSI chip. One usually

needs access to the logical address, but often only the physical address is available outside the chip. Many processors prefetch instructions or data, but may not in fact use all of the prefetched information. This can cause extraneous triggers. Additionally, the addresses (of the type that *can* be accessed) of the instructions or data that are to cause the triggers must be known. The association between virtual and physical addresses may change during execution. The same virtual or physical address used by multiple processes will cause indistinguishable triggers. Moreover, one cannot distinguish between different entities which are sharing code. Of course processes can be "locked" in place in memory during execution, but this is often a gross distortion of normal operation. [RIL87]

To make pattern-matching triggering practical, details of determining patterns to match must be done *for* the user. The user should be allowed to insert measurement pseudo-instructions in the source code, with the system software such as compilers, linkers, and loaders assembling the corresponding trigger patterns. These patterns would then be down-loaded to the measurement hardware.

Triggering on a Sequence of Patterns. False triggers caused by use of the same virtual addresses in multiple tasks can sometimes be reduced if triggering is based on a required *sequence* of patterns. Some kinds of events can only be identified by a sequence of patterns. Sequences will require a substantial increase in the size of the pattern storage memory and its control hardware.

Both Event and Pattern Triggering. A hybrid system with triggering from both inserted measurement code and pattern-matching hardware has many of the advantages of both systems; the easy correlation of data with program execution, association of process and processor, and knowledge of the state of all processors at the time of each measurement trigger.

Timed Trace Support Hardware

There exists a progression of time measurement hardware support techniques in which additional difficulty or cost is rewarded by more accurate or more detailed information.

No Measurement Hardware. If there is a single instruction stream, little machine state, and no external real-time constraints, all measurement can be accomplished by operating system services. Even resource utilization details such as cache-updating information can be captured, since they require operating system intervention. Low level hardware resource information such as bus access latency and data-path occupancy will not be available.

One should note that many SIMD machines fall into this class, wherein special measurement hardware is not needed to obtain useful information. However, detailed measurement of the communication characteristics of SIMD machines may require extensive instrumentation. The addition of another instruction stream, say by use of a separate I/O processor, or use in a real-time application, may preclude use of this no-hardware measurement approach.

Central Time Counter Register. Measurement of MIMD machines requires that at least a time counter be available to each processor. In a shared-memory architecture, this could be a globally-accessible register as illustrated in Figure 1. Loosely-coupled

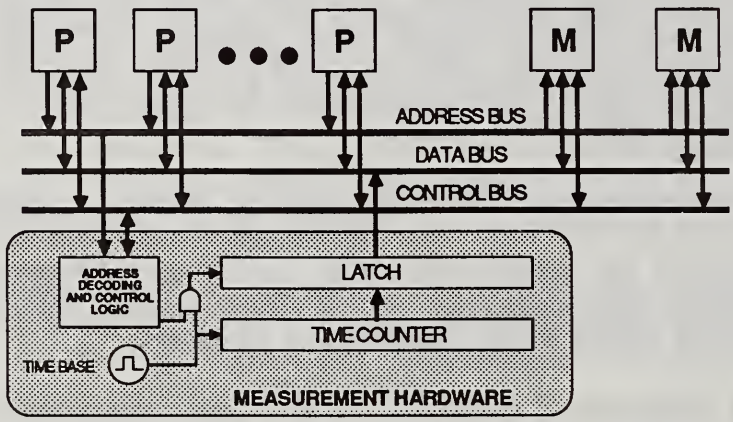


FIGURE 1 - GLOBALLY READABLE TIME COUNTER

systems require some sort of time distribution systems to be discussed below. Each processor could refer to the time register by a simple read instruction, greatly decreasing the perturbation as compared to an operating system call. This technique only allows elapsed-time and process-processor association measurements; data about utilization of resources such as caches, buses, etc., requires the addition of some resource measurement hardware to be discussed later. Once data is obtained by a processor, it must be stored either in local or central memory. The obtaining of time data and the storing of it demand processor cycles on the single processor where the time trigger occurred, tie up the memory path for all processors and may perturb register, cache and memory management state. We understand that this measurement facility is available on the Cray X-MP.

Built-in Hardware - A counter with adequate time resolution, yet a sufficiently long epoch, must be provided somewhere in the address space and implemented as a read-only I/O or memory location. A length of 32 bits is often a good compromise; four thousand seconds (2**32 microseconds) or 2**32 machine microcycles is an adequate experimental epoch, but 16 bits (65 milliseconds - 64K microcycles) is usually not enough. The counter must correctly handle the situation where it *should* be incremented while it is being read without either losing counts or causing an erroneous reading. A central facilities board would seem to be a likely physical location for the central counter.

Optional Hardware - A time-counter board may be located in a globally-reachable I/O basket. Reading data from I/O space may block the system bus until the read is satisfied, causing extra perturbation. Since many modest systems use standard I/O buses (Multibus (TM), VME (TM), etc.) this may be an especially attractive approach. Alternately, a special time-counter card may be provided to attach to the memory bus instead of a memory module. This method will often use less bus-time, but the memory bus is usually proprietary, and its interface is often expensive. As yet another alternative, a dedicated connector may be provided for

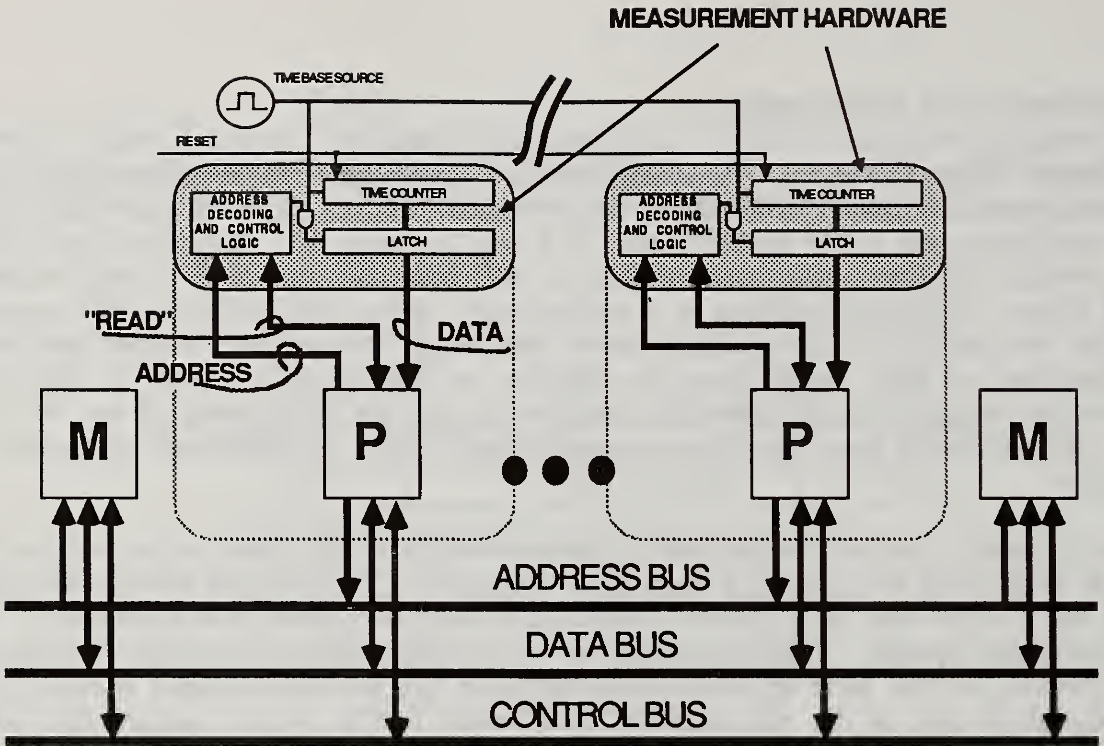
attachment of a time counter.

Distributed Time Counters. Anything less than very tight coupling discourages use of a single central time counter. The extra communication delay to read a central register in machines such as the Butterfly and RP3 may be excessive. Loosely-coupled systems certainly require a time counter local to each processor due to the long communication delays. All such counters in a system have to be synchronized. The counters could be internal to a VLSI microprocessor and would require only a time-base and a synchronizing or reset signal from the outside, as illustrated in Figure 2. The synchronization might be coded into the timebase signal, for the saving of one of these pins. No additional pins would be needed should there be additional processors on each chip.

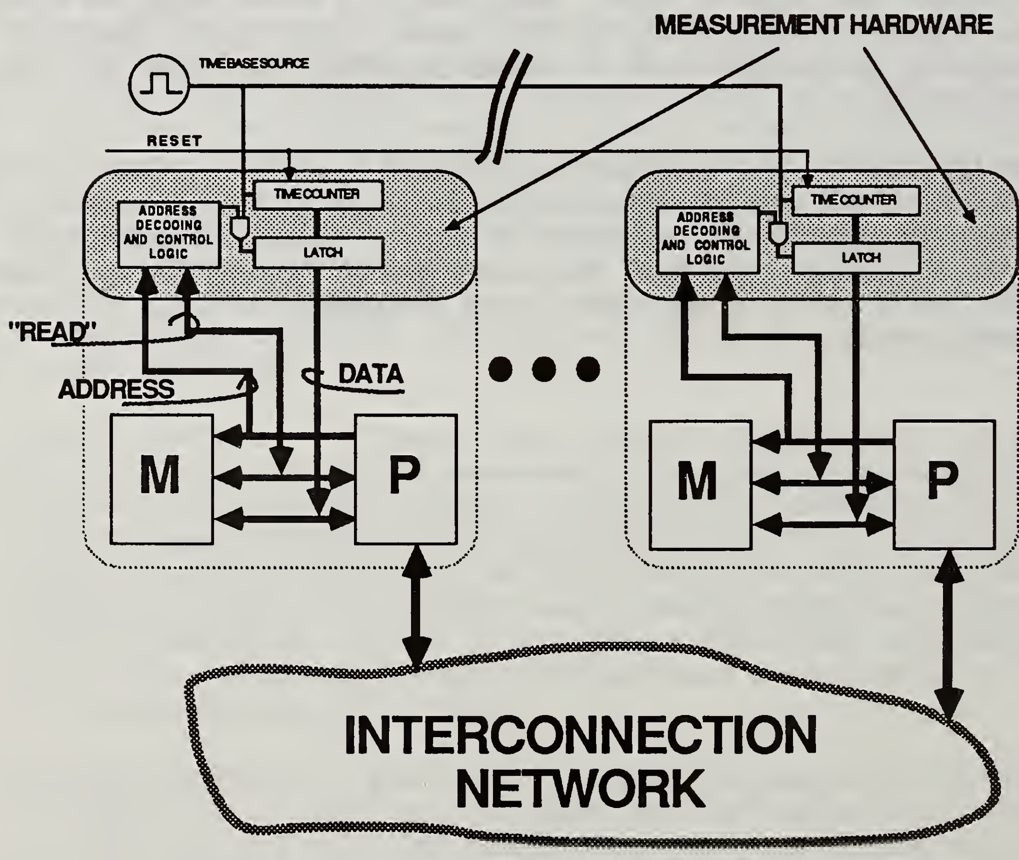
In a loosely-coupled system, all measurement data would have to be collected locally at each node and reported after the experiment. This has the benefit that there would be no extra load on the interconnection network to perturb the experiment. In a tightly-coupled system, distributed time counters would result in reduced perturbation since reading of the time register would not use the normally shared memory data path. Local storage of the measurement data would further reduce perturbation otherwise caused by writing of the event data to central memory during the experiment.

Built-in Hardware - This is a relatively low-cost addition, requiring only a time counter register, and logic to slightly delay any *read* which might be attempted while the register is incrementing (or *vice versa*). This register should be *very* close, logically, to the processor, to avoid perturbing the cache and memory management system.

Optional Hardware - Such a register could also be provided as a low-cost option. The time-base for the time counter should be distributed by always-installed system wiring, but the time-counter register could be on a small daughter board, and mapped as a location in the local processor's I/O or memory address space. It should be connected by a path that avoids disturbing the cache and memory management system.



A - SHARED MEMORY



B - LOOSELY COUPLED

FIGURE 2 - DISTRIBUTED TIME COUNTERS

PERFORMANCE MEASUREMENT APPROACHES			
Event Trigger	Hardware Enhancement	Comment	Resource Utilization Preprocessing
Inserted code.	No measurement hardware.	Only appropriate for SISD and SIMD non-real-time systems.	Can be used. Started, stopped, and read by operating system routine.
	Central time counter register.	Requires globally-accessible memory or input-output. Much less perturbation than above.	Can be used, but causes more perturbation since counters must be read.
	Synchronized local time counters.	Suitable for both loosely- and tightly-coupled systems. Less perturbation.	Can be used; increased perturbation. Should be local to each processor.
	Time counter and global "report" interrupt served by microcode.	Reduces amount of local measurement memory needed. Not for real-time systems.	Same as above.
	As above with global "capture" interrupt served by microcode.	Allows correlation of activity of all processors. Even more perturbation.	Same as above.
	Single central off-machine event data collection system including clock.	Reduced perturbation since clock tags data automatically Not for loosely-coupled systems.	Automatic collection of resource utilization data with no *extra* perturbation.
	As above with global "capture" interrupt served by microcode.	Much perturbation since the captured state must be written to central hardware.	Same as above.
	Off-machine event data collection with head-end for each processor.	Useful on both loosely- and tightly-coupled systems. Doesn't use shared data paths so less perturbation.	Automatic collection of resource utilization data with no *extra* perturbation.
	As above with global "capture" interrupt served by microcode.	Less added perturbation since distributed data collection.	Same as above.
Pattern monitor.	Pattern-matching unit for each processor.	*No* perturbation to execution. Hard to set up. Requires many internal connections. Activity of all processors known at each trigger. Triggering on virtual address often prevents process identification.	Cause *no* perturbation.
	Same, except that a sequence of patterns required for trigger.	Same, except less ambiguous triggering. Even harder to set up.	Same as above.
Hybrid trigger.	Trigger from both inserted-code and patterns.	Inserted-code trigger allows process-address correlation. Pattern trigger causes no perturbation.	Same as above.

Table 1 - Performance Measurement Approaches

Distributed Time Counters and a Global REPORT Command. The above systems suffer from either the perturbation of accessing a *central* measurement data memory area, or the cost of providing substantial measurement data storage memory locally at each processor. Much smaller local measurement data memory could be used (in some cases) if a global REPORT signal (causing a local interrupt at every processor) were used to cause all processors to simultaneously abandon their normal tasks and dump their local measurement data memory to a central memory area. Execution would continue, without distortion to inter-processor timing, when the REPORT signal was removed. Since the processors are not executing user code when servicing the REPORT interrupt, the time-stamp counters must be **stopped** during the the time the REPORT signal is active. This approach would not be useful where external real-time constraints exist; it does, however, maintain the interprocessor timing relationships. If at all practical, the REPORT interrupt should be served by microcode internal to the processing element in order to disturb the machine state (caches, instruction queues, etc) as little as possible, as illustrated in Figure 3.

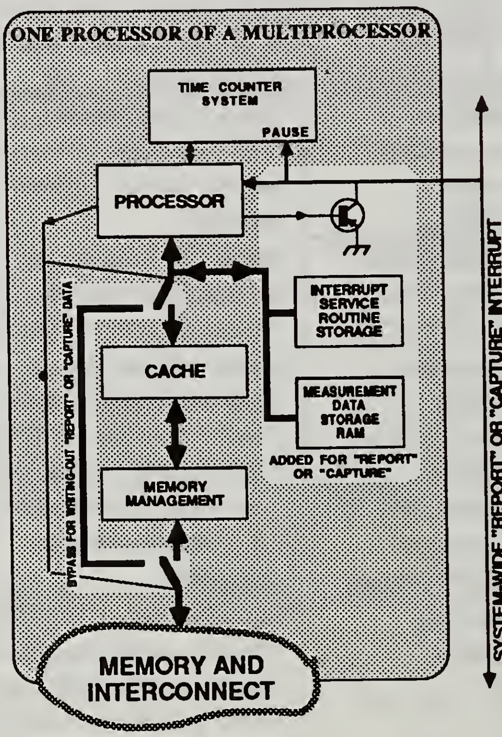


FIGURE 3 - ADDING A "REPORT" OR "CAPTURE" COMMAND

The REPORT command is only meaningful if the individual processors keep their collected data locally, rather than immediately send it to a central location as it is collected. This means, of course, that a small section of very local RAM needs to be provided at each processor.

Built-in Hardware - The REPORT interrupt service routine must be executed from *very* local memory at each processor to minimize perturbation to instruction and data paths, including memory management and cache systems. The service routine might best be in microcode. While violating the goal of simplicity, the data writes caused by the REPORT command should bypass the cache, memory

management, etc., thus avoiding some perturbation to the processing element state.

Optional Hardware - To allow the REPORT facilities to be offered as an option, a fair amount of extra hardware, including means to add redirection of instruction and data accesses to the added local instruction and data memory, would have to be included in all systems manufactured. This might not be an acceptable cost.

Distributed Time Counters and a Global CAPTURE Command. Inserted-code event triggering has the deficiency that only the state of the processor executing the *event* inserted code is captured. A CAPTURE system can be added to overcome this deficiency. If a global CAPTURE interrupt line is provided in the computer system, any processor serving an inserted-code measurement data taking *event* can command all other processors to cease normal activity and save their current process progress and time information. This must be accomplished by "internal" code at each processor to avoid modification of cache contents, etc., as illustrated in Figure 3. One should note that the hardware implementation of the REPORT and CAPTURE commands is identical, only the interrupt service routines would differ. Both commands can exist in the same system, requiring only the second global special interrupt line and interrupt service software, but they can share the added RAM and ROM.

The CAPTURE command wire should be "open collector". Once asserted by any processor, all processors should hold the signal in the asserted state until finished with their capture routine. When the last processor finishes, the CAPTURE line would return to the quiescent state, and all processors would then resume their normal operation. The global capture signal would add one more pin to a VLSI processor package, independent of the number of processors in the package. Since the processor is not executing user code when servicing the CAPTURE interrupt, the time counters must pause the entire time the CAPTURE interrupt line is asserted. For the same reason, the CAPTURE command may be inappropriate in some real-time systems. The perturbation of a CAPTURE command causing all the processors to simultaneously store a single data point locally would be much less than that caused by the REPORT command, where all processors would have to sequentially transfer their full data collection to the central data storage area.

Built-in Hardware - The interrupt service routine must be executed from *very* local memory to minimize perturbation to instruction and data paths, including memory management and cache systems. The service routine might best be in microcode. The CAPTURE command causes data to be captured in every processor in the system; a rather substantial collection. Since all processors have already ceased their normal processing operation, it may be sensible to immediately transfer the data to a central collection point before resuming normal processing, if there are no external real-time constraints. In loosely-coupled systems one may choose to collect all of the data locally for central collection *after* the experiment.

Optional Hardware - As for the REPORT command, a fair amount of extra hardware may have to be included in all systems manufactured in order to allow the CAPTURE facilities to be offered as an option. This might not be an acceptable cost.

Off-Machine Timing Event Data Collection. The code which must be inserted for a time measurement event may be further reduced if each timing measurement *event* consists solely of writing to special data collection hardware addressed as a "memory" or "I/O" location. This external hardware contains the time counter; the mere act of writing data to the collection hardware causes the current time to be captured, along with the written data and other information such as the identification of the writing processor. It is important that the identification of the writing processor be made available to the collection hardware. It must be possible to start and stop data collection and the time counter (to ignore execution of non-user software). Either additional addresses or certain data bits may be used for these functions.

Central Off-Machine Timing Event Data Collection. In a shared-memory architecture, a single central location can be the target of all the system's event data writes. This approach is only suitable for machines with an easily-accessible global memory or I/O area. The factor-of-three-or-more time penalty to reach a global location in switch machines (Butterfly, Ultra, RP3, ...) would make this central-collection approach fairly perturbing. Placing the data collection address in memory space (instead of I/O) usually results in less system perturbation because of more efficient transfer protocols. Each process should identify itself in the data it writes. There should be hardware identification of the processor at each event data write. In a bus-oriented machine, the writing processor should be identified on the machine's bus so that the experimenter will not have to make special internal connections to the machine. User design of processor identification circuitry is usually impractical, given the proprietary

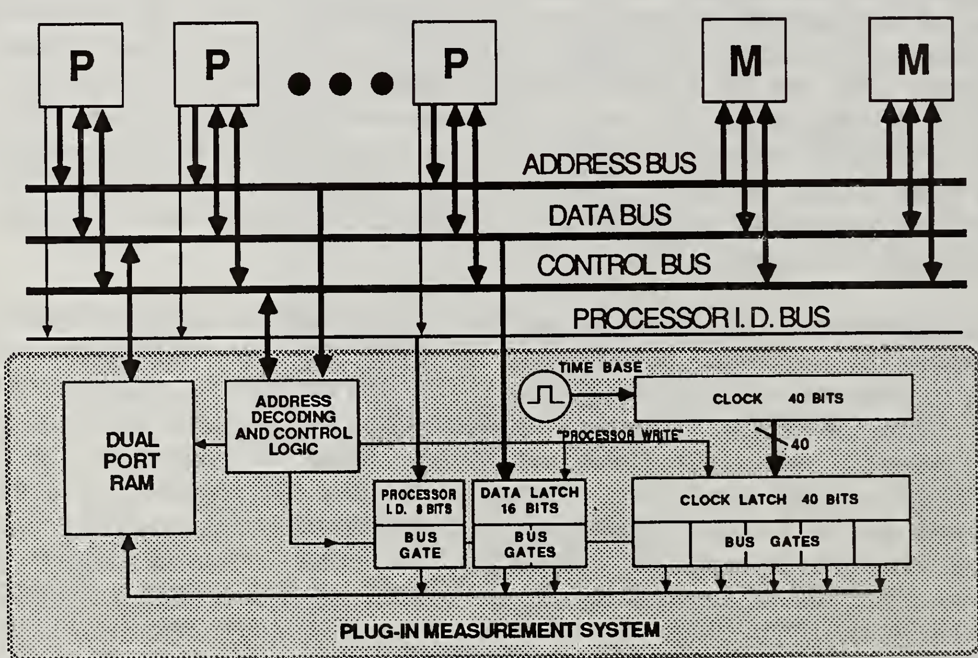


FIGURE 4 - CENTRAL OFF-MACHINE DATA COLLECTION

nature of the details of present-day computer hardware. The NBS Trace Measurement System (TRAMS) [MIN86] [MIN87] is an example of off-machine timing event data collection applied to a commercial multiprocessor. We required very detailed

proprietary hardware information about the multiprocessor computer in order to design the processor identification hardware.

A more integrated revision to TRAMS is under construction. This will incorporate all measurement data storage on the TRAMSII card, which will be readable and configurable from the system under test after the experiment, as illustrated in Figure 4. Thus a single added board serves an entire tightly-coupled shared-memory system. This hardware measurement support requires access to most of the processor-to-memory or I/O signals, or to a system-wide bus.

Built-in Hardware - There is little to be gained by building-in this feature, as long as the writing processor is identified on an accessible high-speed bus.

Optional Hardware - In a bus-oriented machine, it is relatively simple to provide a central data collection system as an option. Providing identification of the writing processor on the system bus is the major expense that would be borne by non-purchasers of the measurement option.

Central Off-Machine Event Data Collection with CAPTURE Command. As mentioned before, measurement systems based on user-inserted *events* do not normally capture the state of other processors in the system at the time of the event. A CAPTURE command can be added to off-machine data collection of event time trace measurement data. Operation would be the same as with the on-machine implementation of the CAPTURE command, except that the data would be immediately written to the central off-machine address - by all processors. One would have to suspend all normal operation until all of the state information from all of the processors had been transferred, probably a lengthy operation. This would not perturb the interprocessor timing relationships, but would be intolerable if external time constraints existed.

The addition of a CAPTURE command to *central* off-machine data collection would not seem to be generally satisfactory. The perturbation would probably be worse than in the case of on-machine data collection with the use of a local time counter and local data storage memory as described above. Use of *distributed* off-machine collection with the CAPTURE command is a much better approach.

Distributed Off-Machine Timing Event Data Collection. In loosely-coupled systems (and in many switch-connected "tightly"-coupled systems), the collection of timing event data must be distributed throughout the system being measured to avoid serious perturbation. A common notion of time must be available to all processors. The communication speed and bandwidth available over the normal loosely-coupled interconnection network to a central point is too low to permit its use for real-time time distribution and performance data collection. One could add a special parallel time and data network. However, it would require at least 33 to 64 interconnection wires to conduct time and measurement information. This network would connect a central counter and data collector and each processor in the system. A more attractive solution involves replication of at least some of the data capture hardware (including a synchronized time counter) at each processor. Each such measurement "headend" should include a FIFO buffer to smooth out the peak rate of the collected measurement data and allow serial transmission of the captured data back to a central collector. This approach is illustrated in Figure 5. As in other distributed-counter systems discussed above, only one or two global wires are sufficient to synchronize and operate

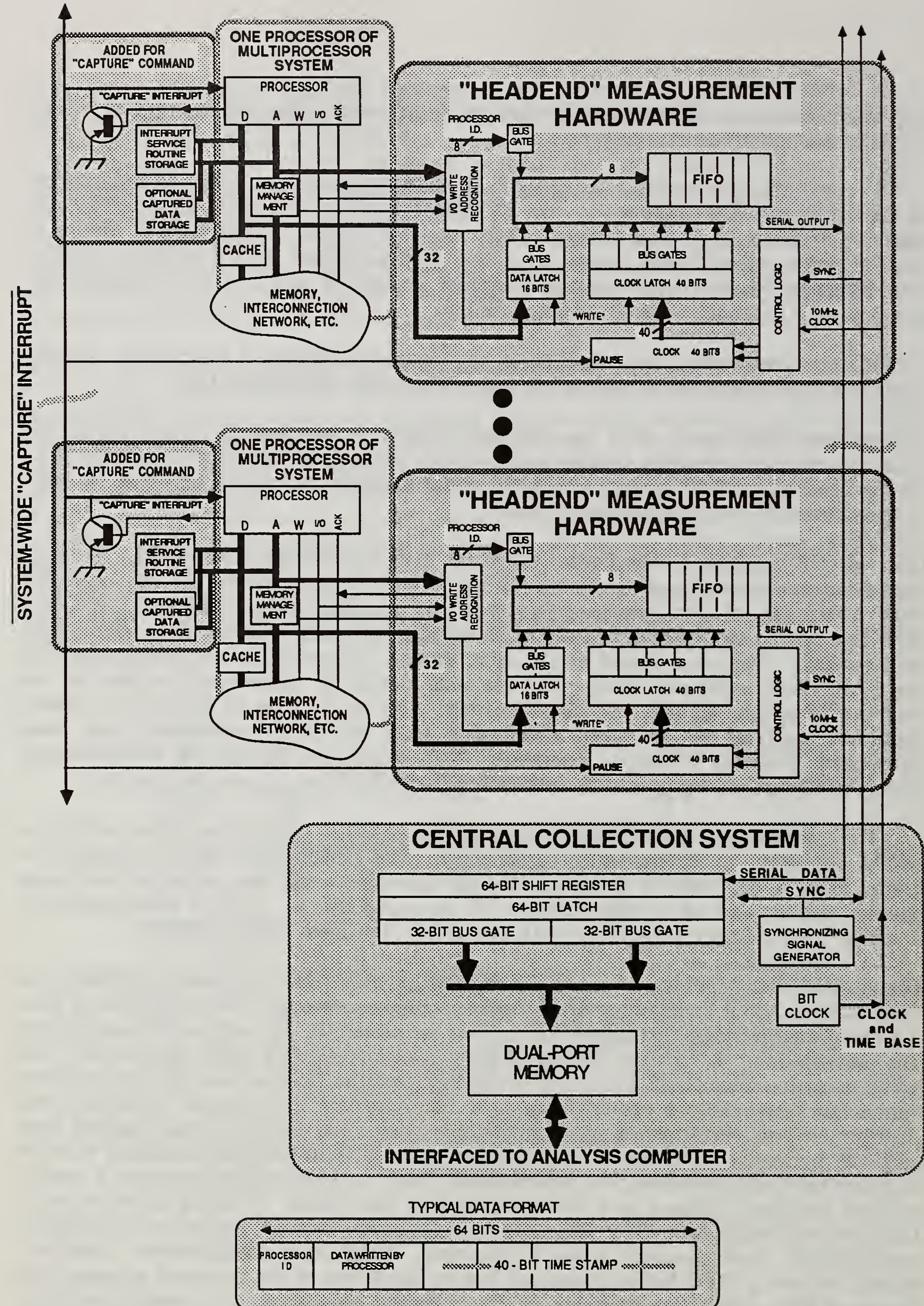


FIGURE 5 - DISTRIBUTED OFF-MACHINE DATA COLLECTION WITH "CAPTURE" COMMAND

the time counters. Another global wire is needed for the serial data transmission back to the central collector. Access to this common serial return path must be arbitrated by a suitable local area network medium access algorithm, (perhaps the simple collision-free algorithm proposed for ANSI X3T9.5 LDDI [CAR84] and used in the Digital Equipment Corporation VAX Cluster(TM) network).

The headend portion of the system which must be replicated for each processor is not large. It could either be an application specific integrated circuit (ASIC) - perhaps gate array - or might be incorporated within a VLSI processor itself. Only one central collection system is required in each system. One should note, however, that there is substantial traffic over the serial link whereby data is returned to the central collector. At 64 bits per measurement event, and a transmission rate of 32 megabits per second, events could not be accepted more often than each two or three microseconds anywhere within the processors feeding a single central front-end shift register and dual-port memory. An additional central front-end shift register and dual-port memory would thus have to be provided for each 10 to 50 processors, depending on the size of program "granules" being time traced.

Distributed off-machine data collection can also reduce measurement perturbation in tightly coupled machines, since it avoids use of the computer's shared data path for collection of measurement data.

Built-in Hardware - The measurement headend associated with each processor requires only two or three external signal leads, if it is incorporated on the VLSI chip with the processor. This same number of connections can serve several processors located on a single VLSI chip. Much of the headend would not have to be replicated for the individual processors on a VLSI chip containing multiple processors.

Optional Hardware - It is practical to offer this type of measurement hardware as a piggy-back option at each processor. At least the clock, synchronization, and serial returned data wires (3 total) should be prewired to every processor site. The processor's memory or I/O bus must be made available for writing of data to the associated measurement headend. The headend should be an ASIC for size and cost reasons.

Distributed Off-Machine Data Collection with CAPTURE Command. The perturbation caused by a CAPTURE command would be reduced by distributed event data collection (compared with central data collection) since the FIFOs in the return paths would allow the processors to write their state data without having to wait for the shared data path. The upper left hand corner of Figure 5 illustrates the additions needed to add the CAPTURE command to a distributed off-machine collection system. Of course the time-stamp counters must be **stopped** during the the time the CAPTURE signal is active. Again, the CAPTURE command might cause too much perturbation to be used in systems with external timing constraints.

Built-in Hardware - The addition of a CAPTURE command always involves major modification of the data paths in or near the the processor itself, as illustrated at the upper left in Figure 5. A special interrupt must be added, and it is desirable to be able to bypass the normal instruction and data paths. There would be much less impact on the design of the central off-machine collection hardware, except that

much more data would be collected. This substantially-increased traffic over the serial data link to the central collector reduces the number of processors which could be served by each central front-end shift register and dual-port memory.

Optional Hardware - The CAPTURE command has little effect on the add-on distributed measurement hardware, except that the timing clock must be stopped during CAPTURE interrupt service. Since it has such a major impact on the processor itself or the close-by data paths, either it would have to be included in all processors, or as a special processor version.

Non-Perturbing Triggering. All of the above time event triggering systems require execution of "measurement code" which perturbs the operation of the system being measured. On the other hand, recognition of time trigger *events* can be based on passive monitoring of the addresses or values of instructions and data, along with the state of the processors.

Non-Perturbing Pattern-Matching Timing Event Triggering. The trigger addresses or values can be detected by matching the information on the address or data lines with stored patterns. Because of at-processor caching, or lack of a single central monitoring point, this trigger hardware must usually be replicated for **each** processor in the system. A number of different patterns, one for each event, must be matched. In a system with 32-bit matching, one cannot afford to be able to match *all* possible combinations (some four billion), nor would one need to. Breaking the 32 bits into groups the size of the address space of high speed static random access memories (RAM) allows a fully-programmable matcher that can match any desired subset containing many combinations; more can be matched if there are a number of common subterms. What results is effectively the recoding of a sparsely populated state space into a small, manageable state space.

The general outline of a pattern-matching event trigger detection system is illustrated in Figure 6. Instead of basing the match on a single logical product of all 32 inputs, a multistage approach with simpler products to generate intermediate terms is sufficiently flexible and more economical. The 32 input signals (where the input patterns appear) are first separated into groups of 10, 11, and 11 bits. These groups of bits are used to address random access memories (RAMs). The contents of locations corresponding to trigger events contain unique 7- or 8-bit codes (a minimum of 127 unique event codes). These codes are further combined to address the second rank of RAMs. The uniquely encoded locations in this second rank produce two 8-bit event codes. The sixteen bits from the second rank of RAMs address the last rank. There is one RAM in the last rank for each *type* of trigger (global trigger, selective trigger 1, selective trigger 2, ...). Ones are loaded in the last rank of RAMs at addresses corresponding to trigger events. The accumulated RAM access delay may become excessive at some point in the above process. One must then insert a pipelining register at this point - for example between the second and last ranks of RAMs in Figure 6.

System software must be provided to allow the experimenter to define trigger points as pseudo-instructions; these would eventually generate the patterns to be down-loaded to the matching RAMs. This measurement setup software must select non-colliding intermediate terms for the matching process. More details can be obtained from the report on the NBS REMS system by Nacht [NAC87].

Once the timing event trigger has been created, both the trigger pattern (or the encoded event identification) and a time counter value must be captured. This trigger can be used with on- and off-machine data collection in most of the same ways as inserted-code triggers. The REMS system captures data in a distributed, off-machine manner. The *global* trigger from any pattern matcher causes data to be captured at all processors. Actual "trigger" patterns are captured, rather than encoded events, since the full patterns present the easiest way of determining the execution progress at the processors which did not cause the trigger event. Only a single time-stamp is needed in the REMS systems since data from all processors is captured at the same time. Use of the "selective" triggers will be covered in the discussion of resource utilization measurement using preprocessors.

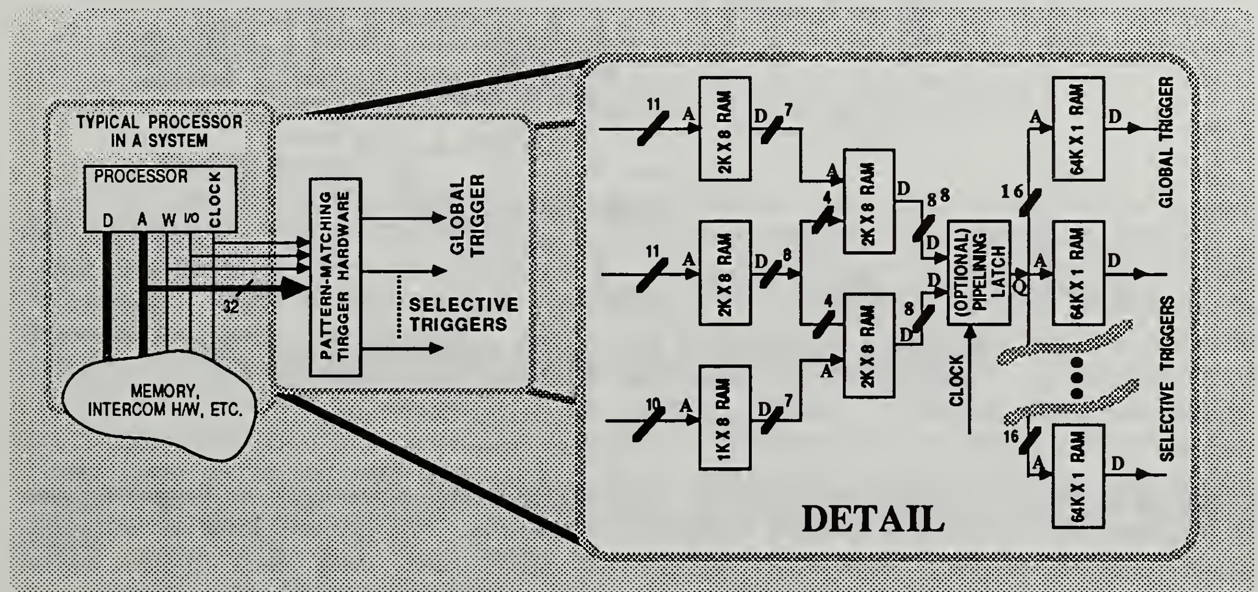


FIGURE 6 - PATTERN-MATCHING TRIGGERING

Because of the cost and the difficulty in obtaining unambiguous event identification, pattern-matching triggering will probably never be attractive alone.

Built-in Hardware - While a pattern-matching trigger mechanism could be incorporated with each processor, this may require *more* external connections than would an off-board implementation. Not only would all of the resulting trigger signals need to lead out, but the pattern itself would have to be available for capture in order to identify the trigger cause. In addition some means would have to be provided to load the matcher RAMs. This system is quite complex and would occupy a lot of real estate.

Optional Hardware - All of the address (or other) lines of interest, along with timing signals, have to be made available to add-on hardware. This requires a substantial interconnect path. Some way must be provided to load the patterns into match storage RAMs from the measurement control computer. Logical address are often not visible from outside VLSI processors. One possible answer to the

problem of inaccessible signals in VLSI circuits is to provide an additional connector on the **top** of the IC package, and to bring out the extra measurement signals to it. This is a variant of an idea used by Zilog. They provided a version of their Z8 single-chip microprocessor with a top socket for an industry-standard EPROM. The EPROM took the place of the normal internal mask-programmed ROM during the software development process and the processor would still mount in the normal Z8 socket.

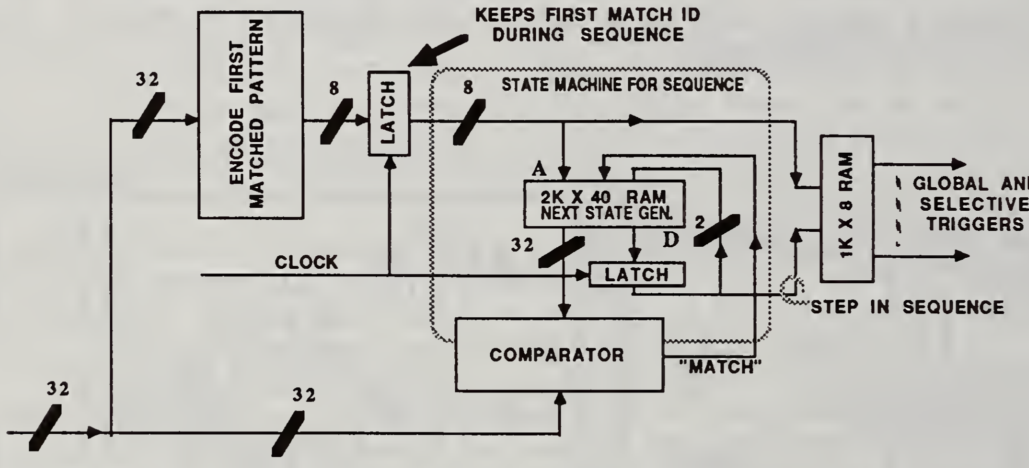


FIGURE 7 - SEQUENCE MATCHING TRIGGERING SYSTEM
(SEQUENCE LENGTH OF 4, BRANCHING FACTOR OF 1)

Triggering on a Sequence of Patterns. Some additional selectivity in triggering can be obtained by requiring a specific sequence of patterns. The general case of recognizing a sequence of patterns is very expensive to implement. Any "first" match would be followed by a tree of state machines of arbitrary branching width at each layer. A great simplification results if the branch width is limited to **one**; no branching is allowed beyond the first match, as illustrated in Figure 7. In this case each "first" match may only lead to a single correct sequence of patterns. The resulting loss in flexibility may be acceptable.

If **M** different sequences of length **N** (or less) were allowed, then an added sequence matching memory of **M*(N-1)** locations would be required. Each location in sequence RAM would need a word-width equal to the matching word length - say 32 bits. A pattern-matching system like that in Figure 7 is used to detect the first trigger. Its output is a pointer to the remaining sequence which must be followed exactly in order to recognize and event. Each of these following steps in the sequence merely requires comparison of the incoming pattern with a single correct pattern from the "sequence" RAM. This is a state machine which must be traversed in the proper order to create a trigger.

Built-in Hardware - As in the case of matching simple patterns, incorporating the matching system on the processor board may actually *increase* the number of external connections required by the measurement system. This system would occupy even more real estate than the simple pattern matching system, but might be well worth it.

Optional Hardware - All of the address (or other) lines of interest, along with timing signals, have to be made available to add-on hardware. This requires a substantial interconnect path, but no more than the simple matcher.

Both Event and Pattern Triggering. A hybrid system with triggering from both inserted *event* code and passive pattern-matching hardware has many of the advantages of both systems: the easy correlation of data with program execution, association of process and processor, etc. Since the execution address of all processors can be captured at each trigger in a passive monitoring system, the state of all processors at the time of each measurement trigger is captured, even when the triggers come from the inserted-code route.

MEASURING RESOURCE UTILIZATION

Once one learns the time required for execution, one wants to know the reasons that execution required this duration. One must measure the utilization loading of many of the component resources of the computer system in order to identify the bottlenecks. Duty cycles must be measured, for example the degree to which the processor-memory path is busy. Pulse widths must be measured, for example the duration that processors have to wait for data or instruction fetches. What fraction of the memory accesses are satisfied by the caching system? Learning these answers requires the addition of resource measurement "preprocessing" hardware, otherwise data would have to be captured for each system microcycle, which is clearly impractical. Roberts [ROB85] has discussed a wide range of possible resource utilization measurements.

Often one is satisfied to learn average values taken over short sections of the test program. Many parameters are actually the ratio of two counts. Real-time calculation of the ratio, on-the-fly, would seem to be too expensive. As a compromise, average values can be obtained by collecting both the numerator and denominator counts and later doing the division during the measurement data analysis processing.

To obtain average values, one counter (A) counts "all" of a class of events, while the other counter (B) counts a subset of the events. For example, the loading on a backplane is measured by counting all backplane time slots in counter A and just the occupied time slots in counter B. For another example, average backplane access delay (at a given processor) can be measured by counting all access attempts in counter B and summing (in counter A) the backplane time slots or processor cycles that occurred during all the waits for access. The ratio A/B is the average access delay. The values in counters A and B can be read at timing events triggers. If the preprocessor ratio counters are reset after reading, the measurement of the resource can be resolved to different periods in the test routine execution without reduced precision. Certain measurement *events* should generate *selective triggers* which cause preprocessor counter resetting.

While accumulation of *average* values of these parameters may be sufficient, capturing the distribution of the individual data items in 'buckets' by ranges of values may be much more useful. One may also wish to characterize memory activity according to selected address regions (hot spots).

Resource utilization measurement hardware can be used in conjunction with nearly any method of detecting the user-specified events to be time-tagged and captured. There is a roughly linear relationship between cost and the number of resources that can be measured simultaneously. A number of pairs of counters would be required to measure all of the items of interest in a multiprocessor system. Since resource measurement hardware is generally non-perturbing, in many cases it can be reconnected to measure other resources and the test stimulus software can then be rerun.

It is vital that all preprocessors be disabled when the system under test is *not* executing test code. No data should be taken when the system is executing the overhead of "measurement" calls to the operating system, etc. Special measurement events can be used to start and stop the data accumulation.

Resource Monitoring with Inserted-Code Triggering. The means of generating data collection triggers is relatively unimportant. Even operating-system measurement calls can be used. The operating system would then manage the preprocessor counters - albeit with considerable extraneous effect on their values due to the operating system's own code. Inserted-code event triggering, using off-machine data collection hardware, can use resource measurement preprocessing very effectively. Specific event codes, or bits, can be used to reset certain preprocessors, giving all the resource-measurement flexibility of the NBS REMS system mentioned below.

Resource Monitoring with Non-Perturbing Triggering. The **global** triggers from a a pattern-matching trigger system cause collection of the current state of all of the resource utilization counter data. In addition, any global trigger can be accompanied by one or more ***selective triggers***, which reset certain preprocessor counter pairs after their data has been captured. The NBS Resource Monitoring System (REMS) [NAC87], in its basic form, is an example of this class of system.

Preprocessor Implementation

Present computer systems pay scant attention to providing access to the "test points" needed for resource utilization measurement. One of the greatest problems that has been observed at NBS in using external measurement systems is that vital status signals are not explicitly and simultaneously present because the designer has chosen logical minimization or reduced pinouts to reduce costs. It then becomes necessary for the measurement hardware to collect a number of signals and perform combinational and sequential logical operations to derive the signal to be measured. The signals to be combined often occur at different times. This makes the measurement hardware very special-purpose, and requires a lot of costly study of usually-proprietary system details before the measurement apparatus can be modified and attached.

The computer designer should consider the types of signals which are likely to be of interest for resource measurement, and arrange logic so that these signals are explicitly present. Status signals are most useful if they are continually present and only unstable at specified times, or always present in a certain relation to the clock phase or system state. The signals should either be present at the time the event recognition system produces its output, or a clearly stated number of clock periods earlier or later - allowing simple pipelining. If the needed signals are thus made available, the interface with the measurement equipment is extremely straightforward, and can be the same for many systems. This could allow performance measurement equipment to become a commodity product, usable by people with a wide range of technical skills.

Ratio Counter Preprocessors. The ratio counters are conceptionally numerator (A) and denominator (B) counters. For some parameters such as duty cycles, the B counter will count *every* bus cycle; once each 5 to 500 ns. In order to allow a reasonable interval before the counters overflow (between places where measurement data capture events *must* be inserted), a counter of perhaps 32 bits will be needed. Since the value of the ratio is never exactly known before an experiment, the two counters should be of the same size. Capturing data with this precision would require a great deal of measurement data memory. Much of this memory, demanded by the measurement precision, is essentially wasted since experimental *accuracy* in the computer measurement field is rather poor due to all of the unquantifiable perturbations. Why store 32-bit data with 6-bit accuracy?

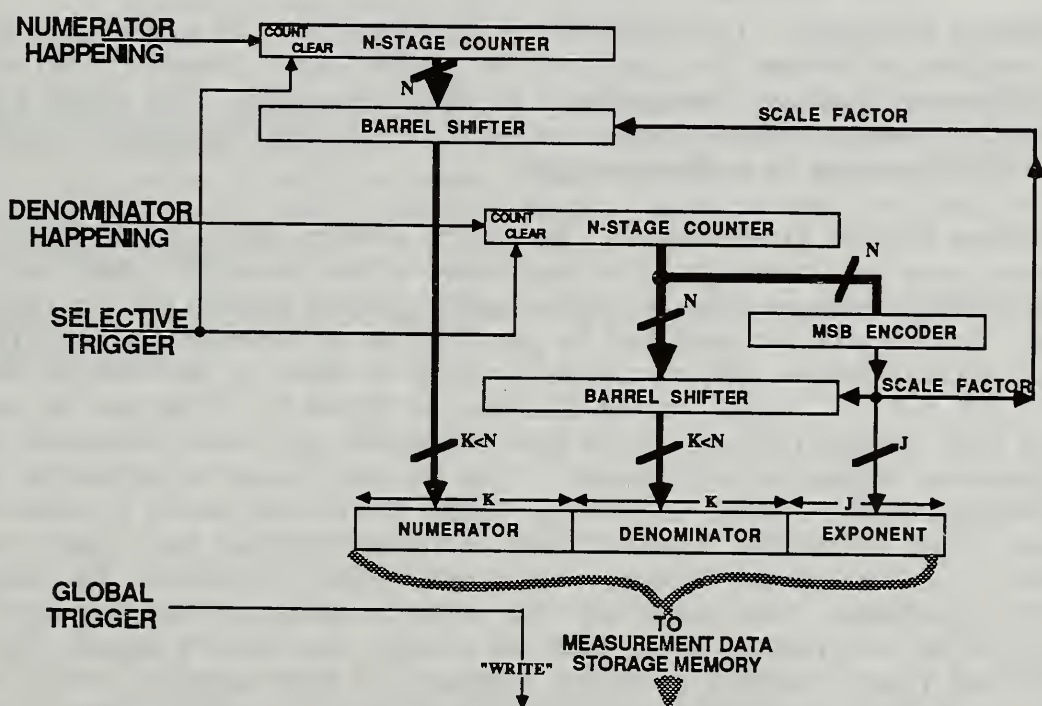


FIGURE 8 - RATIO COUNTER PREPROCESSOR

One is certainly justified in compressing the ratio counter data by use of a reduced-precision floating point format, by employing a self-normalizing ratio counter technique as illustrated in Figure 8. If the measurement is properly designed, the contents of the B counter will always be greater than the A counter, so that *both* counters can be normalized by the same factor as the B counter; use of one common exponent

is sufficient. The main question is whether to compress into 16 or 32 bits.

A counter - 5 bits	B counter - 6 MSB	Exponent - 5 bits

32-bit Ratio Counters Compressed into 16 bits

A counter - 14 bits	B counter - 14 MSB	Exponent - 4 bits

32-bit Ratio Counters Compressed into 32 bits

Compressing into 16 bits limits precision to about three percent of full scale. Compressing into 32 bits allows a precision of about 0.01 percent of full scale. The ratio counter preprocessor of the NBS Resource Monitoring System (REMS), in its basic form, is an example of compression into 16 bits [NAC87].

Built-in Hardware - The cost of providing ratio counters for a vast array of parameters would be great even if they were reduced to VLSI; the cost might be silicon or board area. A more likely approach would be to accept the incremental costs of combining and retiming the signals needed for resource measurement, and to provide logic in the computer-under-test to route selected subsets of the resource signals to a small number of ratio counters. The user could then set up the desired measurements by programming the routing logic to select the subset of interest.

Optional Hardware - One could just as well include, in all systems, the hardware to combine and retime the signals, and to provide logic to route selected subsets of the resource signals to "standardized" interface connectors. This would allow performance measurement equipment to become a commodity product, usable on a number of systems by *ordinary* people.

Distribution Bucket Preprocessors. Short-term average value of resource utilization parameters may not be enough. The distribution of the individual values may also be important for parameters which are pulse widths (access latency, etc.) or memory addresses. This may be accomplished by providing an incrementable register (or measurement memory location) for each possible range-of-values of the variable (for example: <3, 3-5, 5-7, ..., 11-15, >15), as illustrated in Figure 9. At the end of each pulse or upon each memory reference of the type in question (eg. *write, instruction read,....*), the appropriate register is incremented. When the time comes to capture the distribution information, the current value of *all* of the distribution bucket registers must be captured. This information is considerably more detailed than the average ratio data captured in a two-counter system, and correspondingly increases the storage and transfer requirements. One should note that resource utilization data only *need* be capture just before the counters are to reset, not at every time-capture trigger. The collection of data from a number of bucket register, and their resetting, would be time-consuming if done sequentially, and expensive if done in parallel. Since "memory is cheap", one could provide a number of banks of registers by using a large RAM. "Resetting" would be accomplished by moving to a new bank (new area on the memory) to collect data. The experiment would have to terminate when all of the available banks had been used.

The traditional "hardware monitor" as made by Testdata and others provides a mode in which memory activity can be resolved into address ranges. In this case a number of address ranges are established and an incrementable "bucket" register assigned to each range. Additional logic is provided to select only certain types of accesses (instruction reads, data writes, etc., etc.). The same hardware discussed in the previous section can easily be used to provide this function.

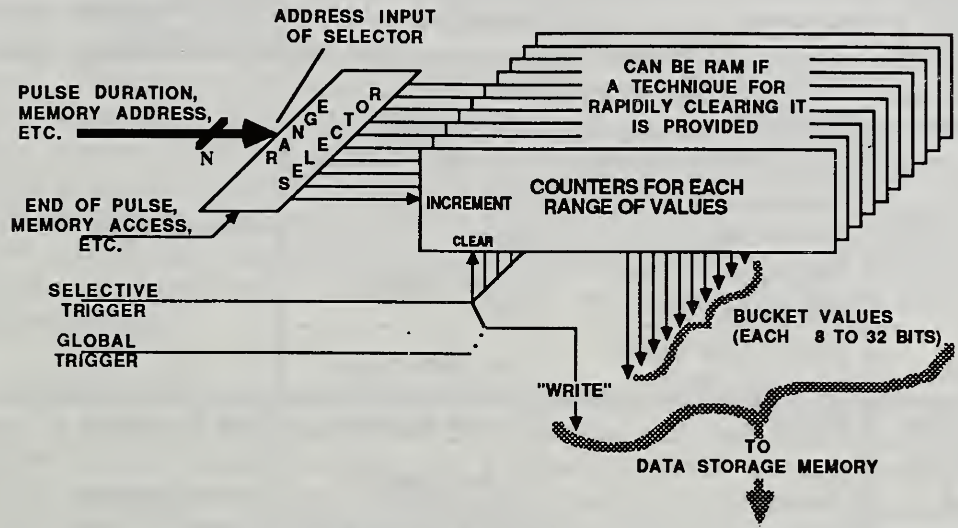


FIGURE 9 - BUCKETS FOR CAPTURING DISTRIBUTION VALUES OF A PARAMETER

A quite different set of attachment points is needed to capture address utilization with a distribution bucket data collection system. Of course all (or nearly all) of the address lines are needed. In addition various status lines must be made available to allow instruction, data, read, write, cache load, memory management table load, etc. actions to be distinguished.

A number of possible preprocessors have been discussed; their applications are summarized in Table 2.

PREPROCESSOR USE			
Parameter	Counters to Obtain Averages		Distribution Value Buckets
	Numerator	Denominator	
Event duration	Number of events	Total duration	Each event's duration
Access latency	Number of accesses	Total duration of all waits	Latency on each access
Memory access distribution	Not Used	Not Used	For each range of addresses
Event ratio	Special events	All events	Not applicable
Cache hit ratio	Cache hits	All accesses	
Bus occupancy	Occupied time elements	All time elements	
Percent time writing	Time spent in write instructions	Total elapsed time	
Percent time writing data	Time spent in write-data instructions	Total elapsed time	
Processor idle	Clock cycles where processor idles	All clock cycles	
Number of events	Not used	Count all events	Not Applicable
Translation look-aside misses	Not used	Count misses	Not Applicable
Transactions		Count transactions	

Table 2 - Preprocessor Use

SYSTEM SOFTWARE ASPECTS

Measurement events must be placed at all entries and exits from the operating system so that user code and operating system characteristics may be separated (cf. [MIT86]). The system software must assure that data collection addresses are always present in the page tables; the delay and perturbation of a page fault caused by measurement reading or writing is unacceptable. As stated above, in order to make use of pattern-matching triggering practical, the details of determining the patterns to match must be done *for* the user. The user should be allowed to insert measurement pseudo-instructions in the source code, and the system software such as compilers, linkers, and loaders should assemble the corresponding trigger patterns. These patterns would then be written in a file for down-loading for setting up the measurement hardware, and to be used in the data interpretation process. It will be impossible for most computer purchasers to make any required measurement changes to the operating system and other system software because of the highly proprietary nature of parallel system source code, and the level of understanding and skill required to make the changes. In any case it makes more economic sense for the measurement "hooks" to be inserted in the system software its supplier.

SUMMARY

The advent of multiprocessor computers has greatly increased the difficulty of designing, choosing, and using computers. Some of these architectures are particularly unsuited to certain algorithms and programming styles. Only through the use of performance measurement can the designer and user obtain the best results (or even acceptable results, in some cases). Performance may vary greatly depending on apparently minor changes in the program or data set, so measurement and tuning are pivotal. Performance measurement of multiprocessor computers can be accomplished over a wide range of cost, accuracy and completeness. While some measurement techniques require hardware support to be built-in by the manufacturer, others can be offered as options with little cost to customers not requiring them, **provided that the hardware design provides access to the required signals**. Accuracy of measurement can be drastically impaired by perturbations caused by the measurement process. Reduction in perturbation increases the cost and complexity of the measurement equipment and process.

Measurement is accomplished in two parts; the establishment and recognition of *events* in program execution, and the collection of facts about the operation of the computer at and between events. As a practical matter, non-perturbing *event* detection is either ambiguous or very complex and costly; it appears that one must tolerate a minimal level of perturbation at each event to achieve an affordable system. This implies that there is a minimum practical granularity in measurement. In MIMD machines, accurate timing and resource utilization collection requires hardware support. There is a roughly linear relationship between cost and the number of resources that can be measured (in detail) simultaneously. In many cases the test stimulus software can be rerun while other resources are be measured, since resource measurement can be non-perturbing.

ACKNOWLEDGEMENTS

This report could not have been written without the many suggestions of my colleagues in the NBS Parallel Processing Group. Particular thanks are due G. Lyon, A. Mink, G. Nacht, and J. Roberts. Useful suggestions were also made by Maj. Mark Pullen, USA, of the Defense Advanced Research Projects Agency.

REFERENCES

CAR84 R. Carpenter, "A comparison of two 'guaranteed' local network access methods", Data Communications, Vol. , February 1984, pp. .

LYO87 G. Lyon, *On Parallel Processing Benchmarks*, NBSIR 87-3580, National

Bureau of Standards, Gaithersburg, MD, Jun 1987.

MIN86 A. Mink, J. Roberts, J. Draper, and R. Carpenter, *Simple Multi-Processor Performance Measurement Techniques and Examples of Their Use*, NBSIR 86-3416, National Bureau of Standards, Gaithersburg, MD, Jul 1986.

MIN87 A. Mink, J. Draper, J. Roberts, and R. Carpenter, *Hardware Assisted Multiprocessor Performance Measurements*, NBSIR 87-3585, National Bureau of Standards, Gaithersburg, MD, Jun 1987.

MIT86 S. Mitchell, *SySM Functional Requirements Description*, Harris Government Information Systems Division, Melbourne, FL, 1986.

NAC87 G. Nacht and A. Mink, *Recommended Instrumentation Approaches for a Shared-Memory Multiprocessor*, NBSIR in preparation, National Bureau of Standards, Gaithersburg, MD, 1987.

RIL87 M. Riley, Telephone conversation regarding how to better accomplish process identification with pattern-matching triggering, Digital Equipment Corp. and Carnegie-Mellon U., 10 Mar 1987.

ROB85 J. Roberts, *Performance Measurement Techniques for Multi-Processor Computers*, NBSIR 85-3296, National Bureau of Standards, Gaithersburg, MD, Feb 1986.

SCH83 G. Schrott and T. Templemeier, "Monitoring of Real Time Systems by a Separate Processor", in *Real Time Programming 1983*, Proceedings of the 12th IFAC/IFIP Workshop, Hatfield, UK, 29-31 March 1983. Pergamon Press, Oxford, UK.

WHI86 J. G. Whitman, Private communication, Harris Government Information Systems Division, Melbourne, FL, May 1986.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO.	2. Performing Organ. Report No.	3. Publication Date
	NBSIR 87-3627		AUGUST 1987

4. TITLE AND SUBTITLE

Performance Measurement Instrumentation for Multiprocessor Computers

5. AUTHOR(S)

Robert J. Carpenter

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The complexity of achieving near-optimum performance from multiprocessor parallel computers demonstrates a need for performance measurement. However, when multiple processors are acting in concert on a single problem, perturbations caused by measurement can be unacceptable. Additional hardware can reduce the perturbation caused by measurement, and can be offered in several stages of refinement and cost. The hardware can often be offered as an option; it is necessary to provide access to the required signals in the system's original design.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

hardware instrumentation; measurement equipment; multiprocessor computers; performance measurement

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

31

15. Price

$11.95

USCOMM-DC 6043-P80